Software Security Models for Service-Oriented Programming (SOP) Platforms

Soutenance de Thèse de doctorat de:

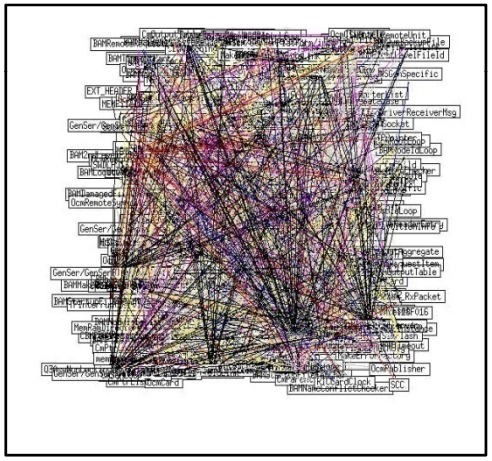
Pierre Parrend
Equipe INRIA-Amazones, Laboratoire Citi
INSA-Lyon
Directeurs de Thèse: Stéphane Ubéda (Pr.)
Stéphane Frénot (McF)

Jury

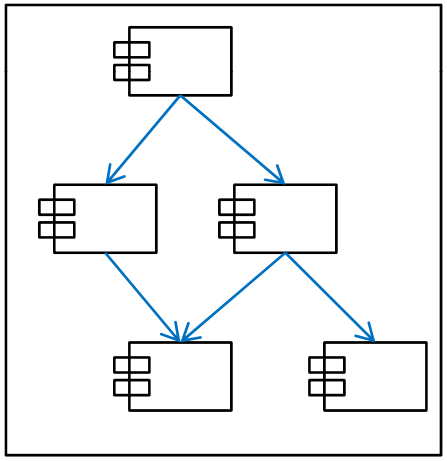
Rapporteurs: Didier Donsez (Pr. Uni. Grenoble I)
Ralf Reusser (Pr. Uni. Karlsruhe)
Examinateurs: Ciaran Bryce (MER Uni Genève)
Pierre-Etienne Moreau (CR INRIA)

Context

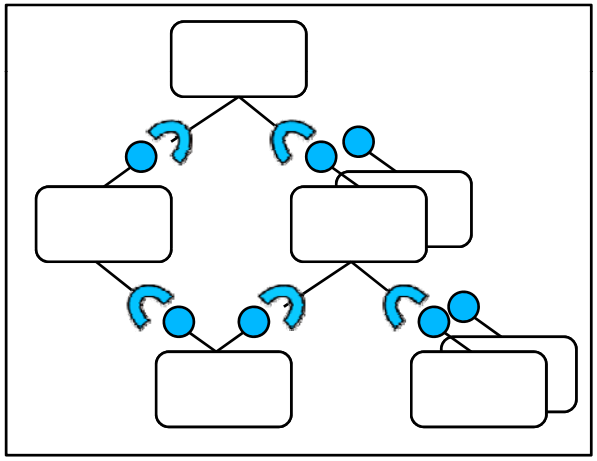
- The Evolution of software



Monolithic Applications



Component-based Applications



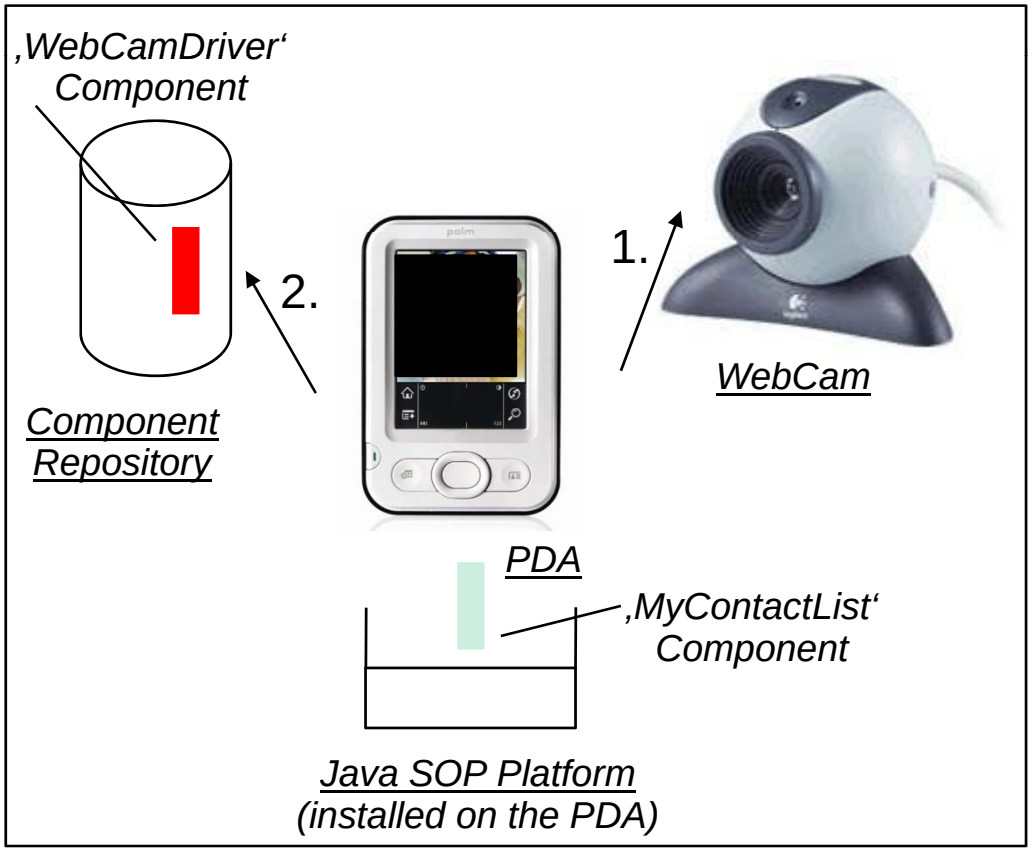
Service-oriented Programming (SOP) Applications

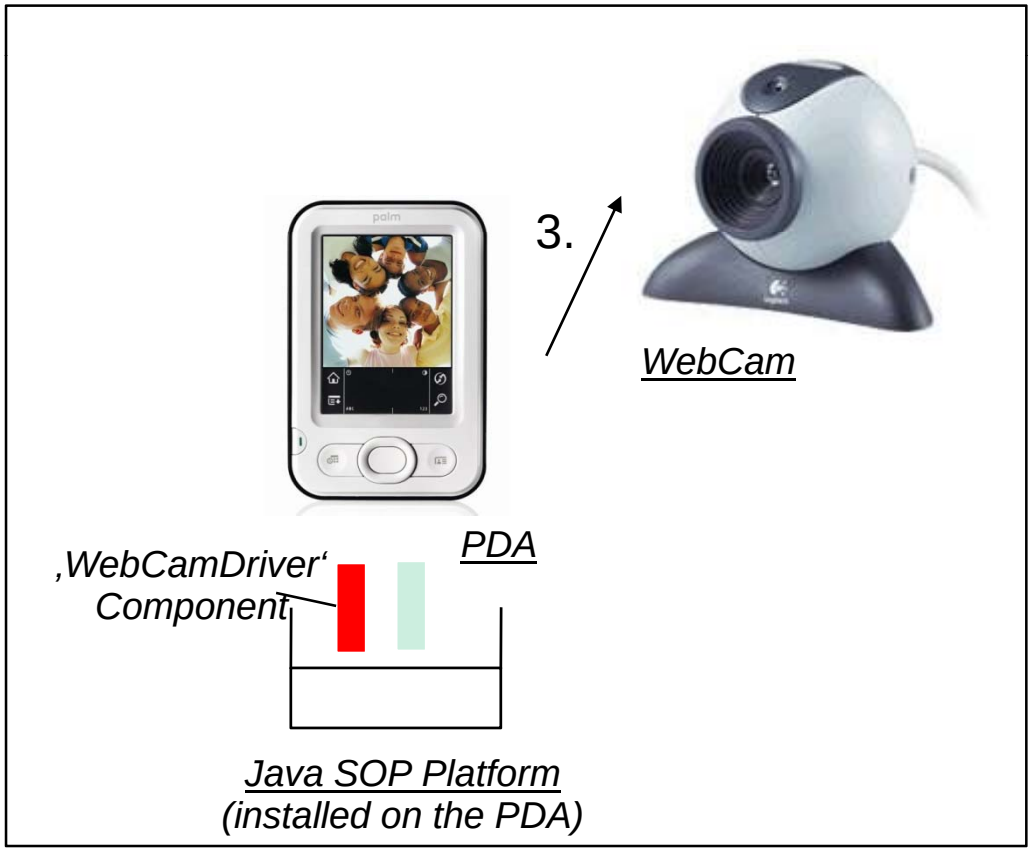
  - Challenges
    - Management
    - Integration
    - **Security**

Environnements d'exécution pour passerelles domestiques, Yvan Royon, PhD Thesis, December 2007.
Spontaneous Integration of Services in Pervasive Environments, Noha Ibrahim, PhD Thesis, September 2008.

Context

- Motivating example: Dynamic SOP applications

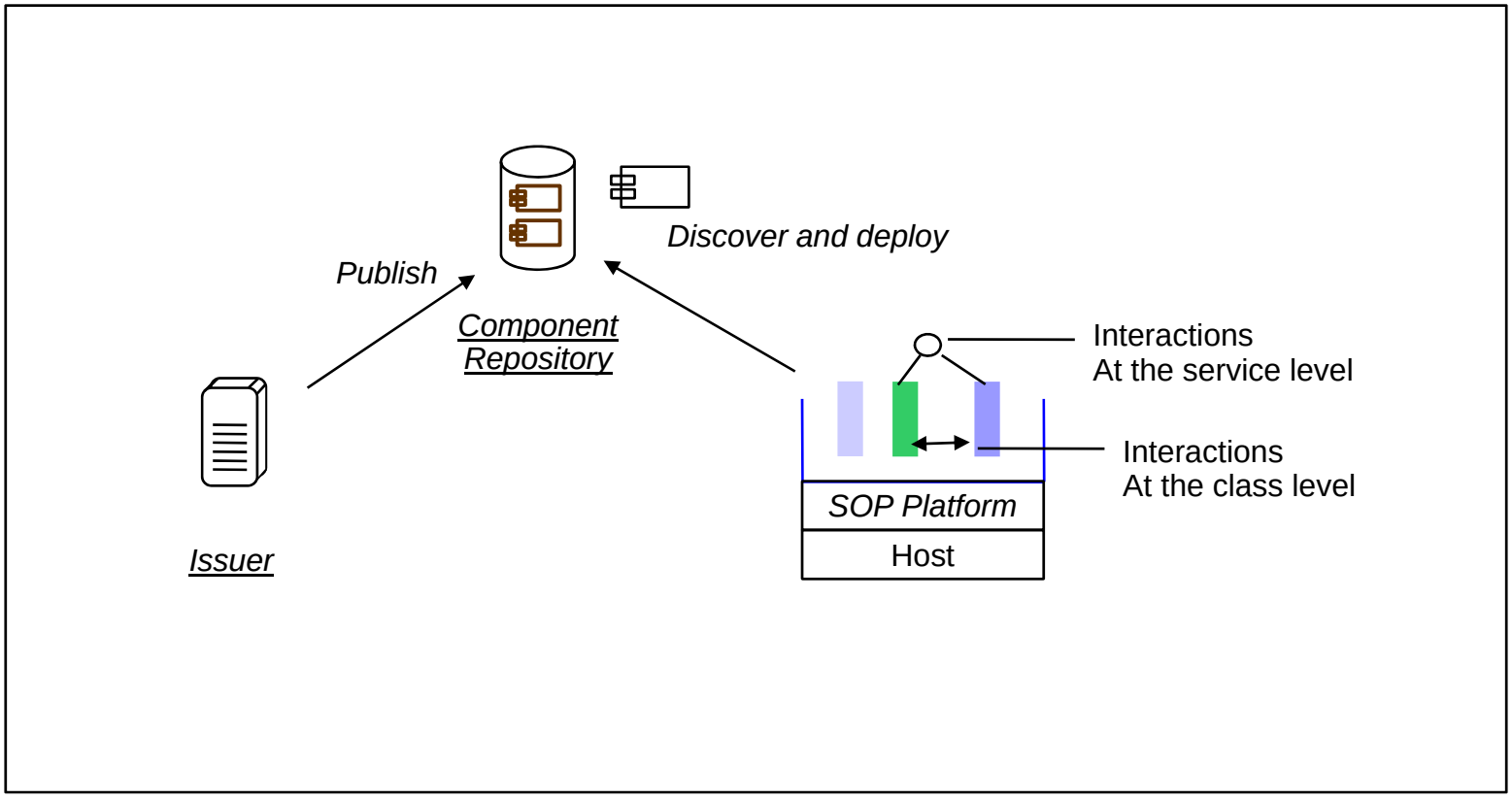




– What happens if the WebCamDriver Component is a Malware ?

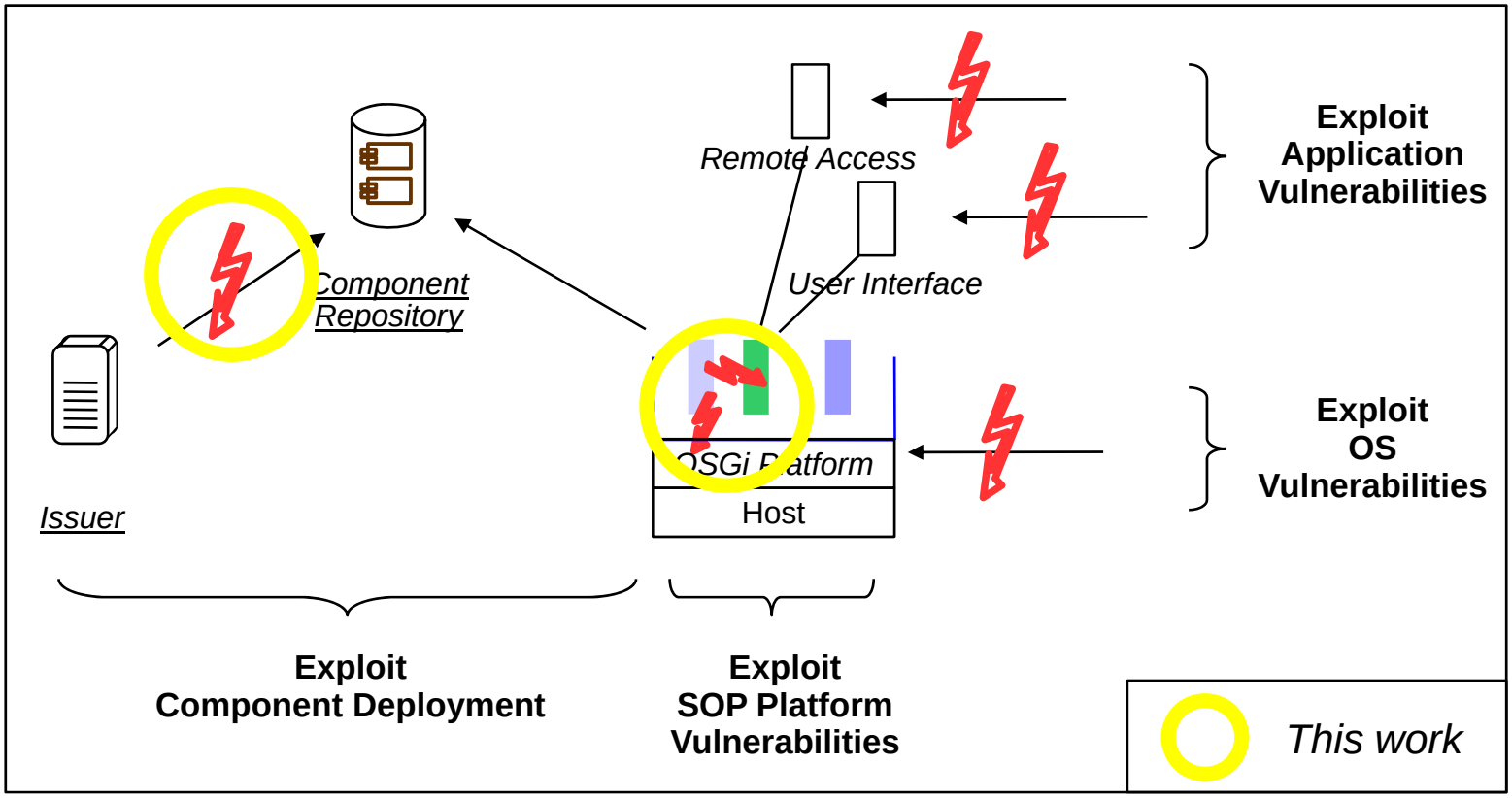
Context

- Service-oriented programming (SOP) platforms
  - EJB 3.0, OSGi, Spring, Google Guice



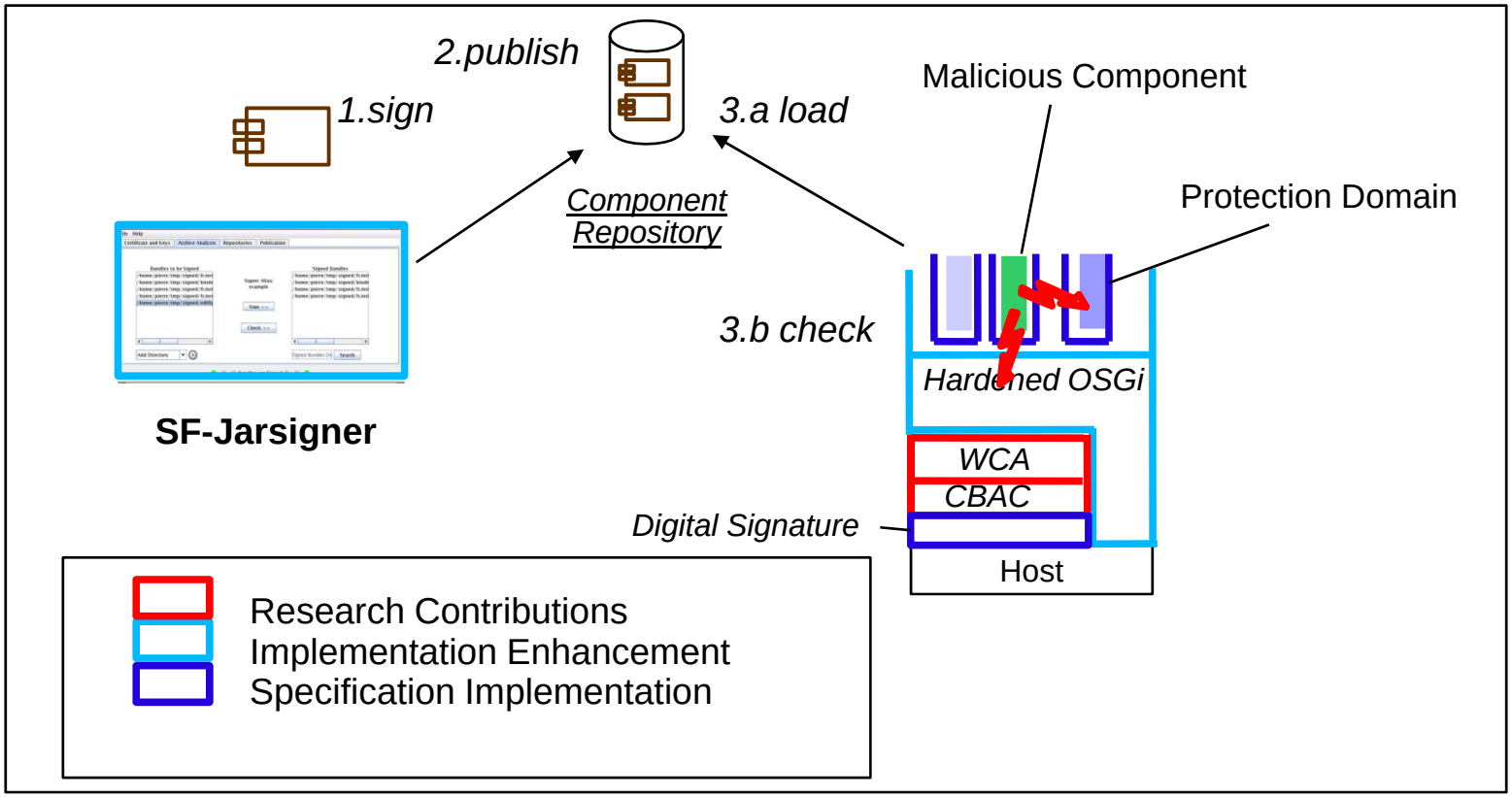
Context

- Attack vectors against SOP platforms
  - Example: The Java/OSGi platform



Context

- Contributions

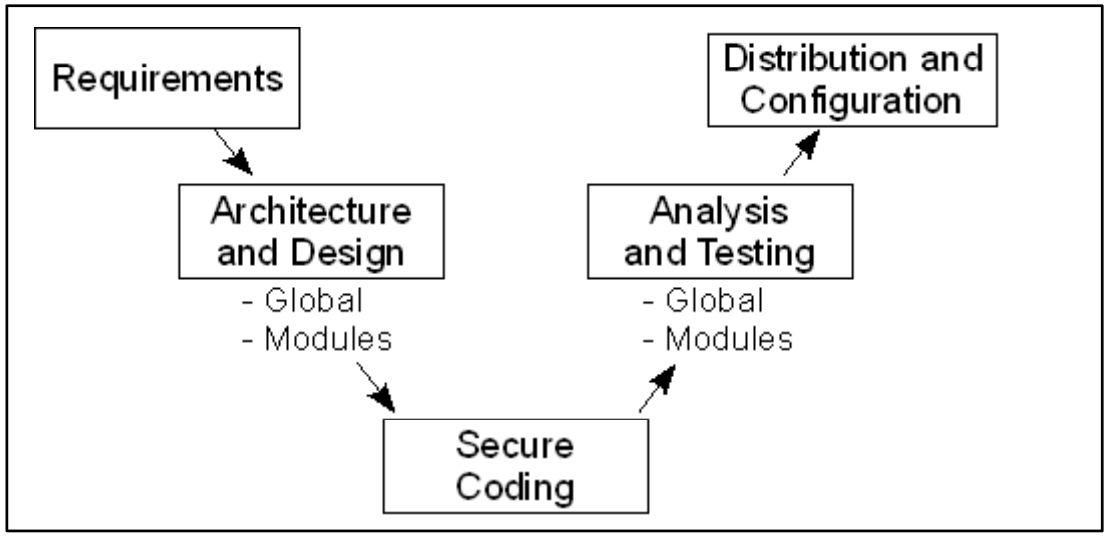


Outline

- **Security for Java-based Software Systems**
- Contributions
- Conclusions

Security for Java-based Software Systems

- Building secure software systems: The software development life-cycle
  - 'Software security assurance'

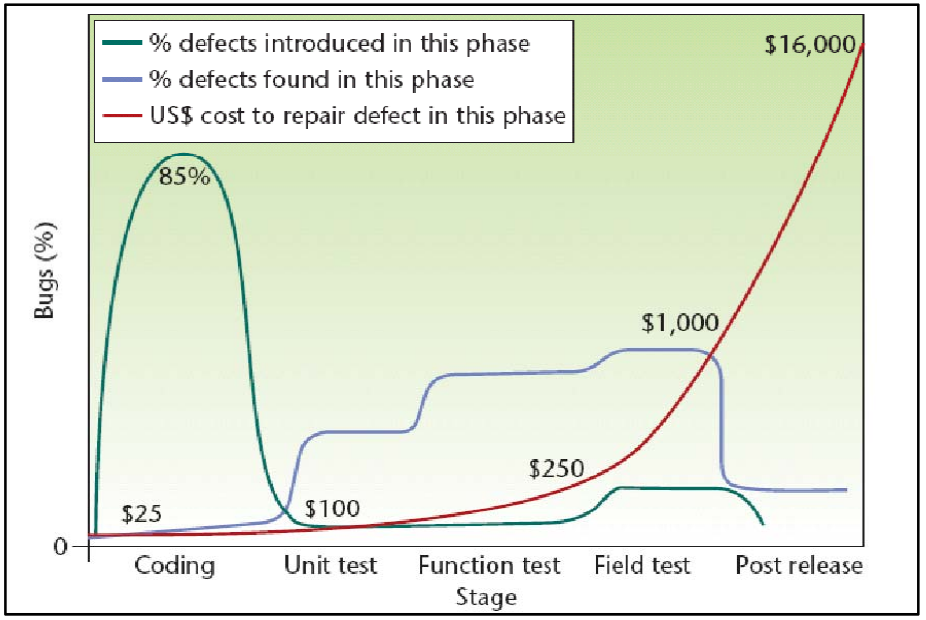


Goertzel, et al. *Software Security Assurance: a State-of-The-Art Report (SOAR), 2007.*

  - Monolithic view
  - Systems are built from several mandatory and optional parts

Security for Java-based Software Systems

- Identification of suitable protection mechanisms
  - Benefits/cost trade-off
  - Cost estimation
    - Minimal when flaws are repared early
    - Grows dramatically latter in the life-cycle
  - Components
    - Reparation only possible if the code is available
    - Detection otherwise



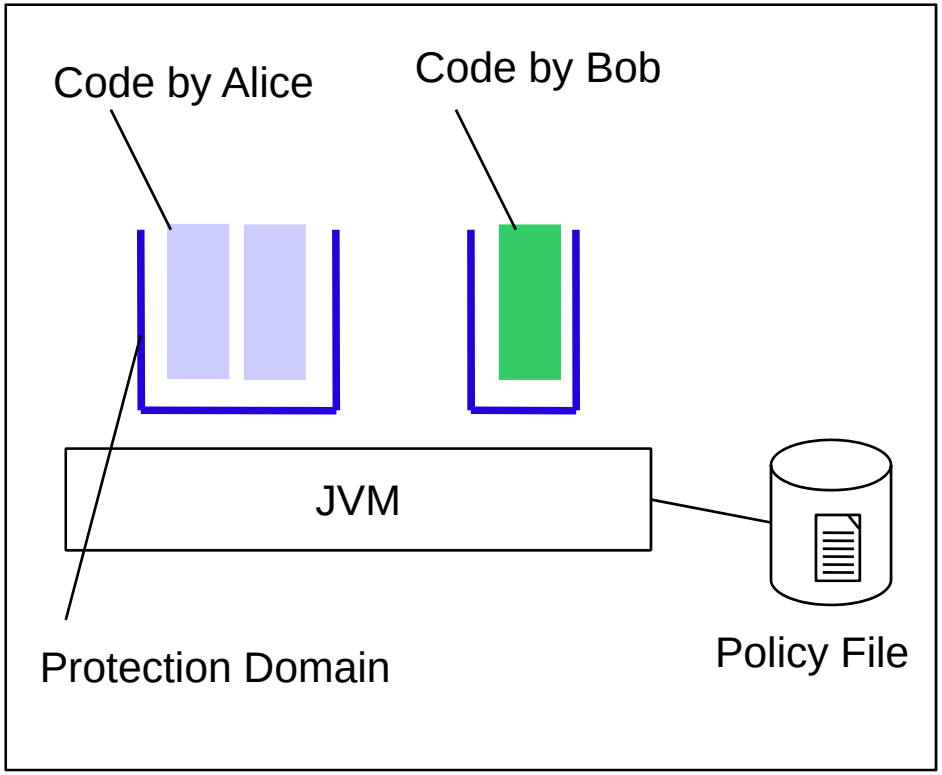
Capers Jones, *Applied software measurement: assuring productivity and quality,* 1999.

Security for Java-based Software Systems

- Java application security: The principles
  - Type safety
    - Objects only perform actions defined through their type
  - Automated memory management
    - Through garbage collection
  - Bytecode validation
    - Executed code is not trusted
  - Isolation of components through class loaders
    - Prevent naming conflicts between components
- Limitations
  - Security use case: execution of one malicious applets in the JVM
  - Class loaders enforce namespace isolation only

Security for Java-based Software Systems

- The Java Security Manager



- Java policy file

```
keystore "file:/home/pierre/keystore.ks";

grant signedBy "alice" {
        permission java.io.FilePermission "/opt/secret/
secretKeys", "read";
        permission org.osgi.framework.Package
Permission "*", "export";
        permission org.osgi.framework.Service
Permission „*"register";
};

grant signedBy "bob" {
        permission org.osgi.framework.Service
Permission "fr.inria.ares.testservice.MyService",
"register";
        permission org.osgi.framework.Package
Permission "*", "export";
};
```

- OSGi: Conditonal Permissions

Security for Java-based Software Systems

- Critics of Java permissions
  - High performance overhead
    - 20 to 30 % runtime overhead
    - Cause the withdrawal of security in commercial applications
  - Hard-coded definition of sensitive methods
    - New permissions for new code only
  - Permission hell
    - Must be extracted for each configuration
    - Tedious manual process
  - Runtime verification
    - Abort or execute dangerous calls
    - In mobile apps for instance, authorization depends on the user

Outline

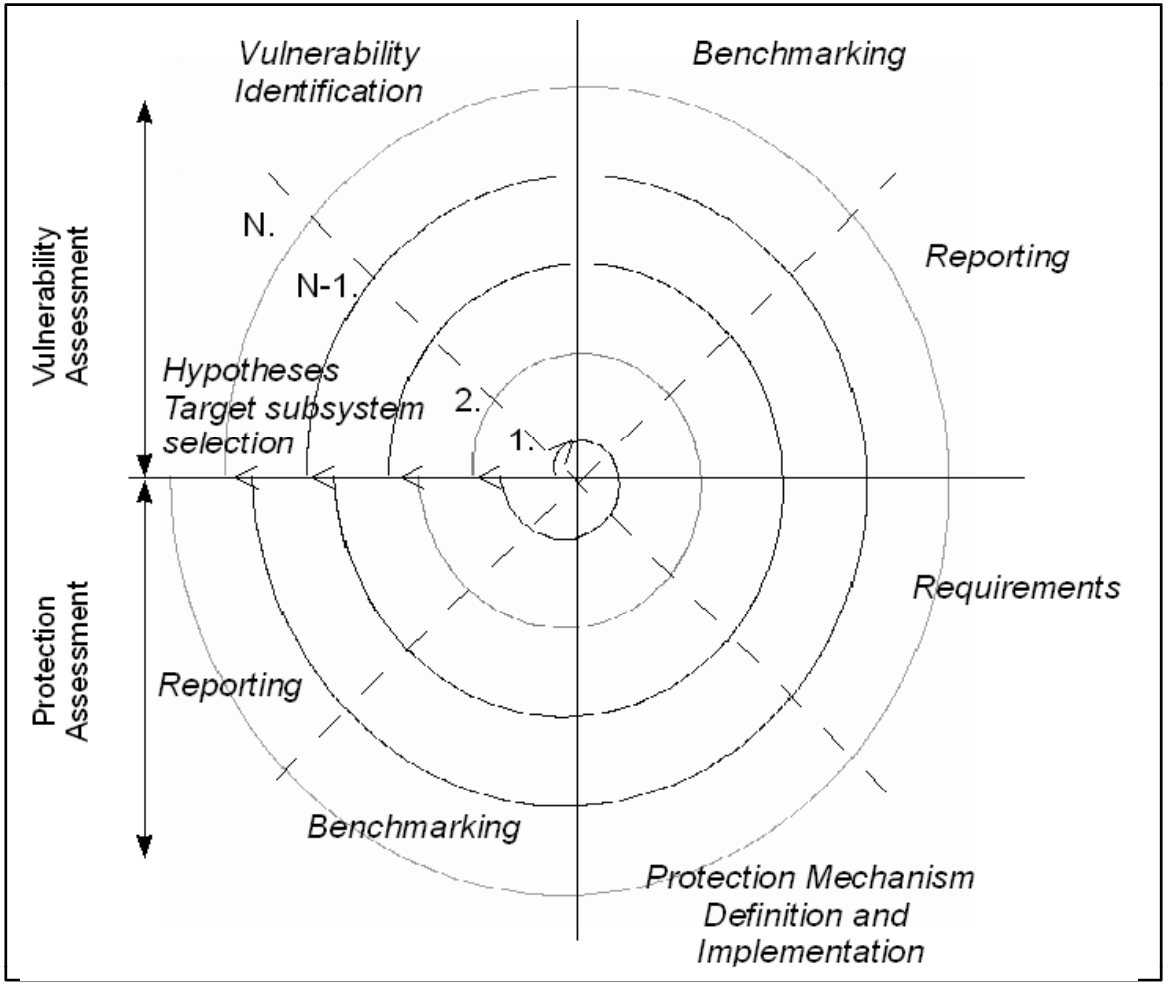
- Security for Java-based Software Systems
- **Contributions**
  - **Building a secure Platform: The *SPIP* Method**
  - Enforcing security for components: CBAC, WCA
- Conclusions

Building a secure Platform: The *SPIP* Method

- The '*S*piral *P*rocess for *I*ntrusion *P*revention'
- The problem
  - Identification of security issues in complex systems
    - For each subsystem
    - Comparison of various implementations
  - Evaluation of protection mechanisms
    - Security assessment
    - Comparison

Building a secure Platform: The *SPIP* Method

- The *SPIP* Method



Building a secure Platform: The *SPIP* Method

- Quantification of the security of a system: the ‘Protection Rate’
  - Security level of complex systems
    - Not a binary metric: never free of vulnerabilities
  - ‘Percentage of the known vulnerabilities that are protected’
    - Against a reference system (here: an OSGi implementation with all known vulnerabilities)
  - Based on the ‘Attack Surface‘ metric *(Measuring Relative Attack Surfaces, Howard, 2005.)*

$$PR = \left(1 - \frac{Attack\ Surface\ of\ the\ evaluated\ System}{Attack\ Surface\ of\ the\ Reference\ System}\right) * 100$$

  - Enables to
    - Assess individual security mechanisms
    - Compare execution environments

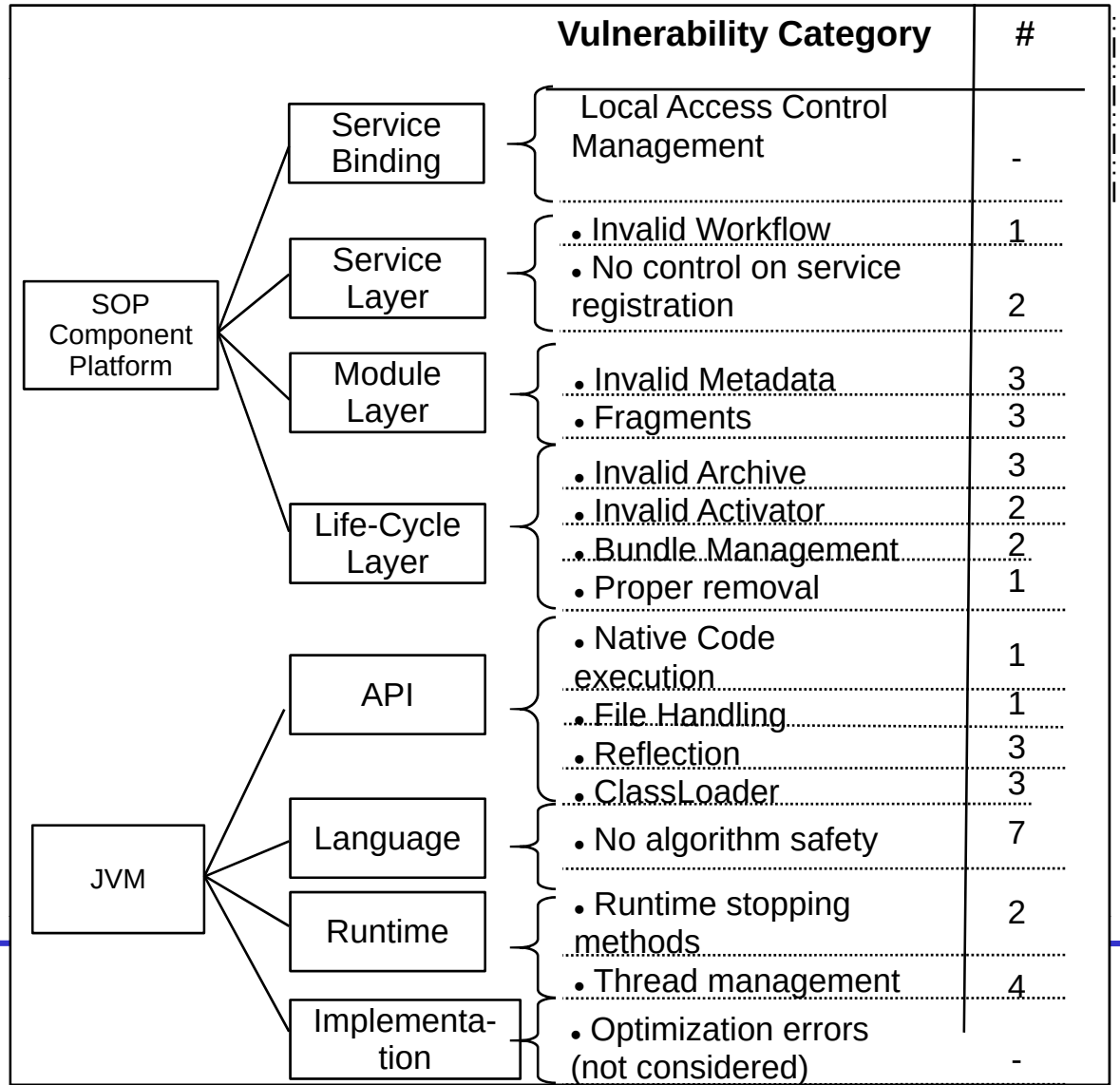
Security Benchmarks of OSGi Platforms: Toward Hardened OSGi, Software, Practice and Experience, 2008.

Building a secure Platform: The *SPIP* Method

- Implementation for the OSGi platform
  - Iteration 1: The Java/OSGi platform
  - Iteration 2 .. 4: Propositions
    - Hardened OSGi
    - Component-based Access Control - CBAC
    - Weak Component Analysis - WCA
  - Iteration 5: Integration with the JnJVM, a secure JVM implementation for OSGi applications

Building a secure Platform: The *SPIP* Method

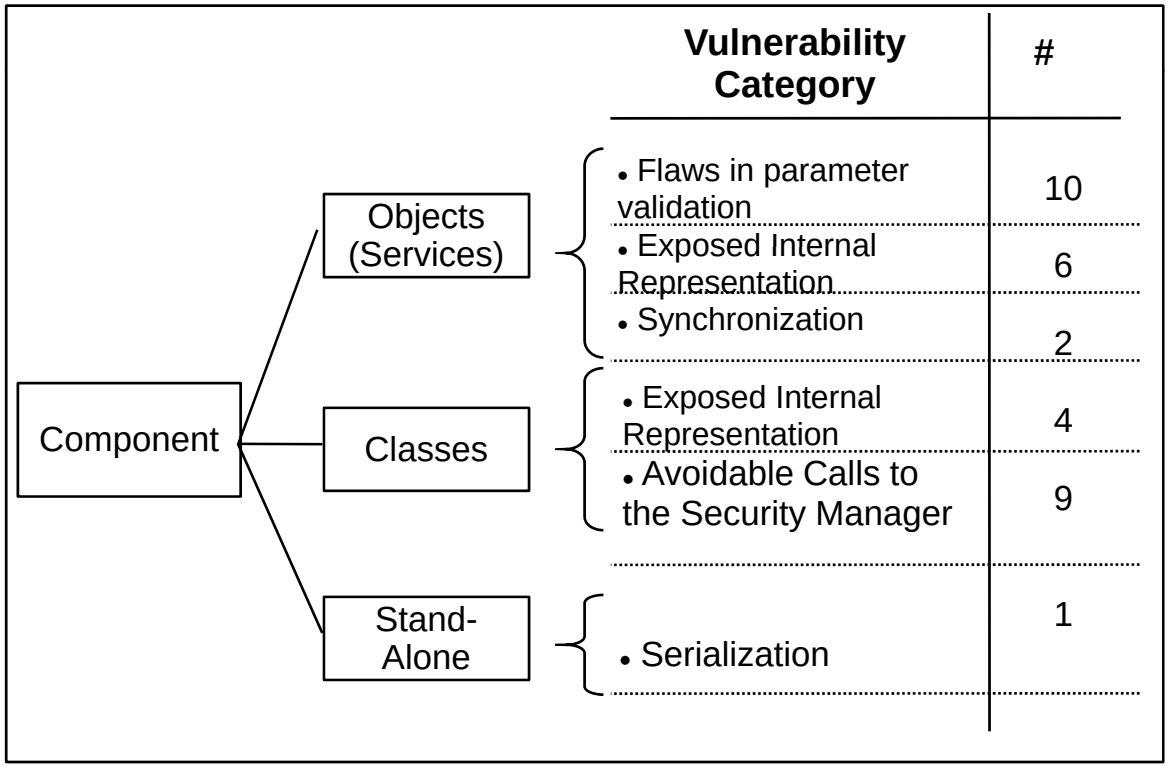
- Results: The vulnerability catalogs – 'Malicious Bundles'



Java Components Vulnerabilities - An Experimental Classification Targeted at the OSGi Platform, INRIA Research Report, 2007.

Building a secure Platform: The *SPIP* Method

- Results: The vulnerability catalogs – 'Vulnerable Bundles'



More Vulnerabilities in the Java/OSGi Platform: a Focus on Bundle Interactions, INRIA Research Report, 2008.

Building a secure Platform: The *SPIP* Method

- Results: ‘Protection Rate’ for mainstream OSGi platforms

Platform Type	# of protected Vulns	# of identified Vulns	Protection Rate
Concierge	0	28	0 %
Felix	1	32	3,1 %
Knopflerfish	1	31	3,2 %
Equinox	4	31	13 %
Java Permissions	13	32	41 %
Concierge with Permissions	10	28	36 %
Felix with Permissions	14	32	44 %
Knopflerfish with Permissions	14	31	44 %
Equinox with Permissions	17	31	55 %

Building a secure Platform: The *SPIP* Method

- Results: Hardened OSGi
  - Protection Rate: 25 % for the 'Malicious Bundles' catalog entries

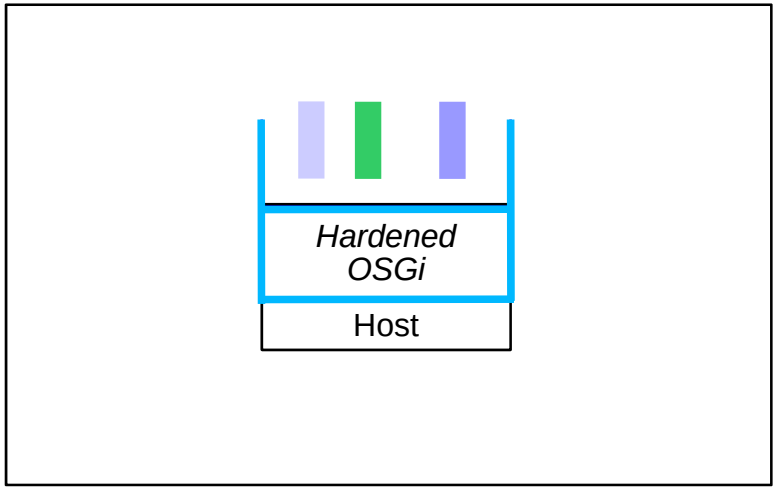
Security Benchmarks of OSGi Platforms: Toward Hardened OSGi, Software, Practice and Experience, 2008.

Introduce

- *Check component size before download, and control the cumulated size of loaded components*
- *Check digital signature at install time*
- *Launch the component activator in a separate Thread*
- *Limit the number of registered services*

Systematize

- *Do not reject harmless unnecessary metadata*
- *Remove all component data from disk at uninstallation*



Outline

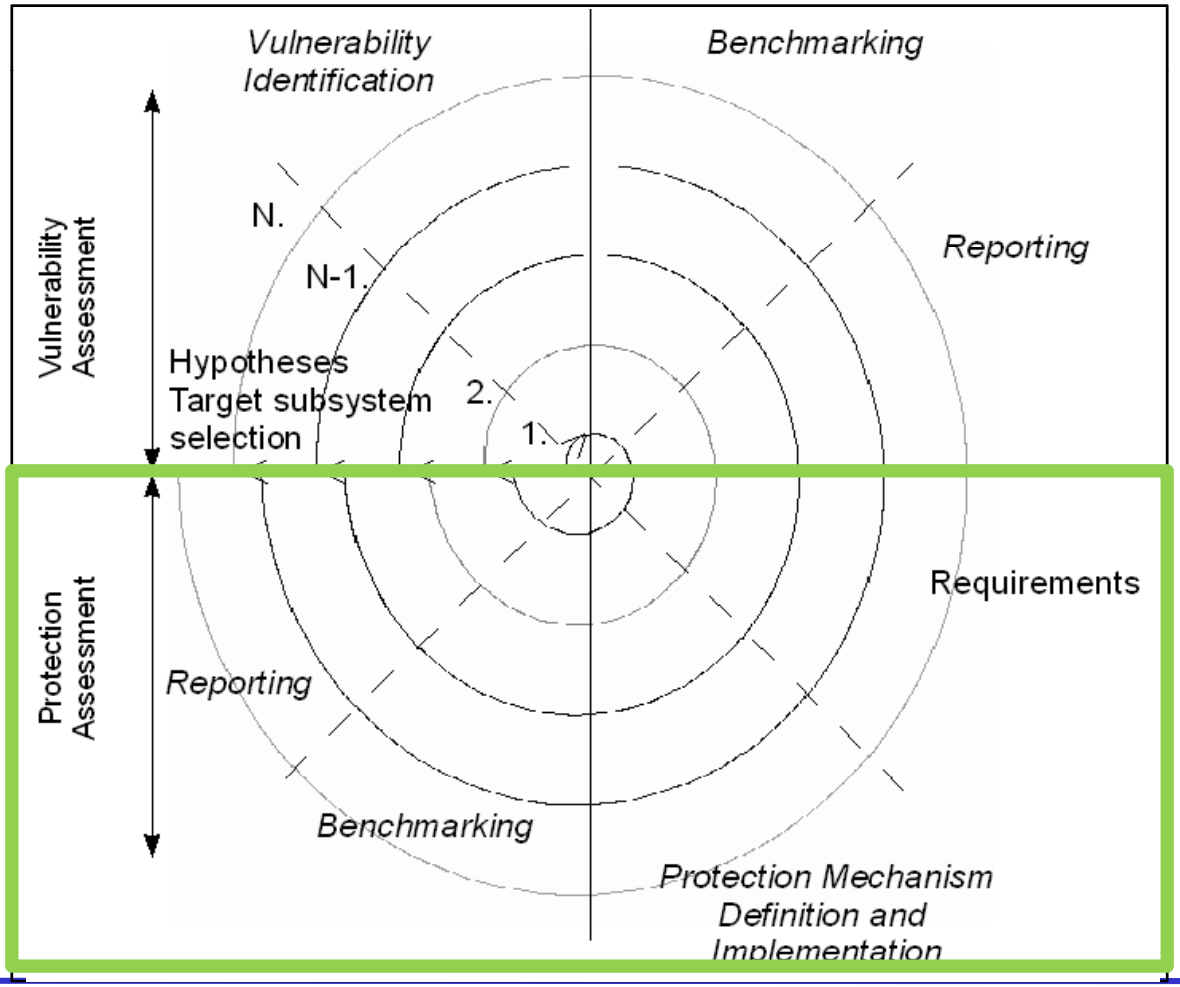
- Security for Java-based Software Systems
- **Contributions**
  - Building a secure Platform: The *SPIP* Method
  - **Enforcing security for components: CBAC, WCA**
- Conclusions

Enforcing Security for Components: CBAC, WCA

- The problem
  - Security issues with components
    - Maliciousness
    - Vulnerability
  - Installing secure components
    - Bytecode analysis only

Enforcing Security for Components: CBAC, WCA

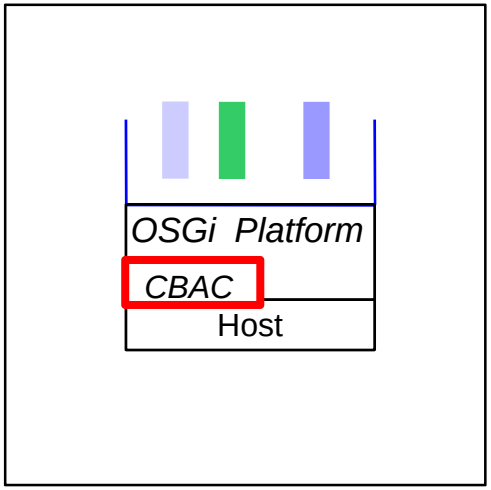
- Definition of tools in the *SPIP* method



Enforcing Security for Components: CBAC, WCA

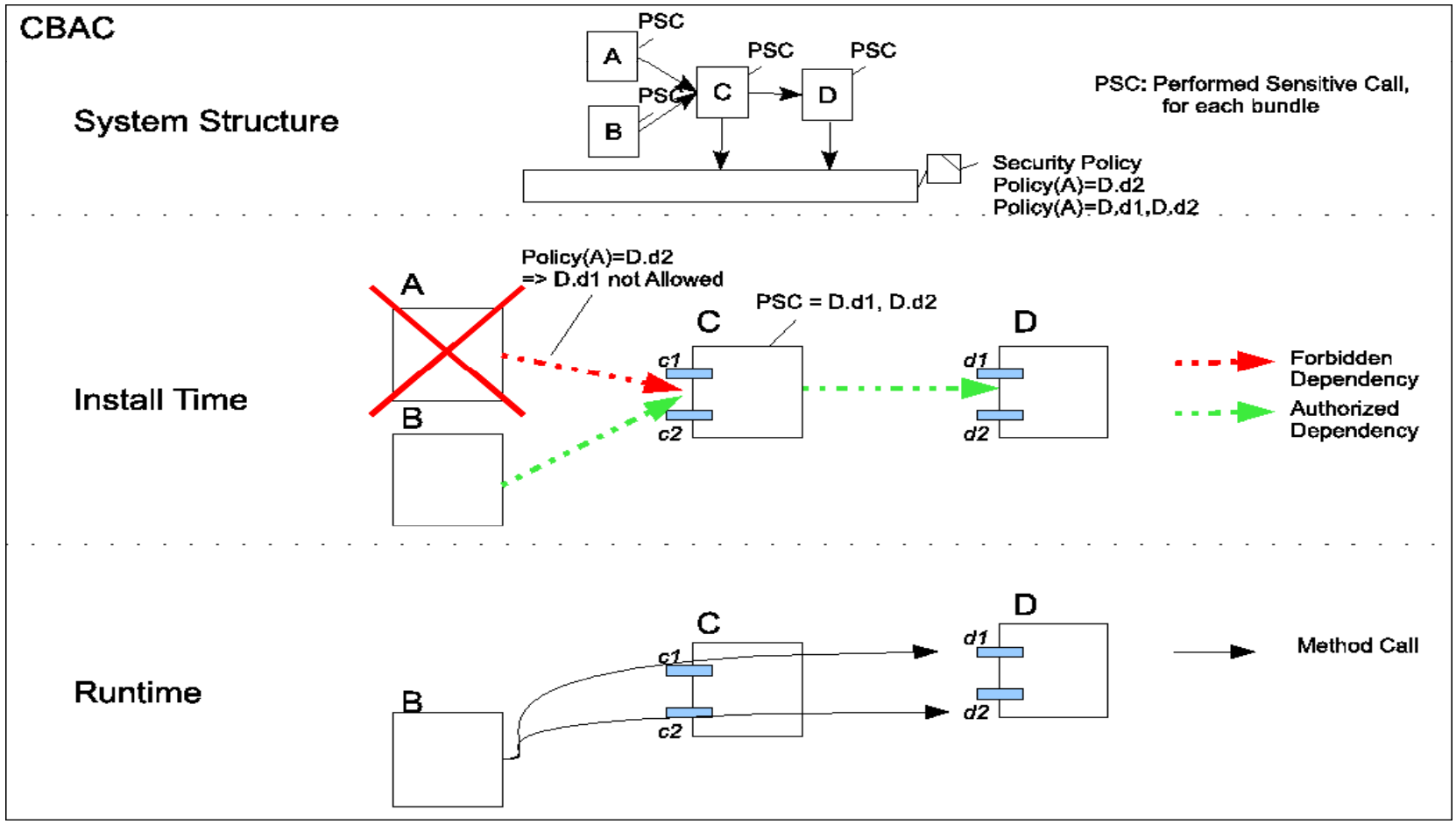
- The CBAC model: Principles
  - **C**omponent-**b**ased **A**ccess **C**ontrol
  - Goal
    - Prevent issues from the ‘Malicious Bundles’ catalog
  - Principles
    - Install time analysis of the execution rights of components
      - Sensitive calls must be explicitly granted
    - Take composition into account
    - Intends to be an alternative to Java permissions
  - Hypotheses
    - The component platform is not modified
    - Each component contains a valid digital signature

Component-based Access Control: Secure Software Composition through Static Analysis, Software Composition, 2008.



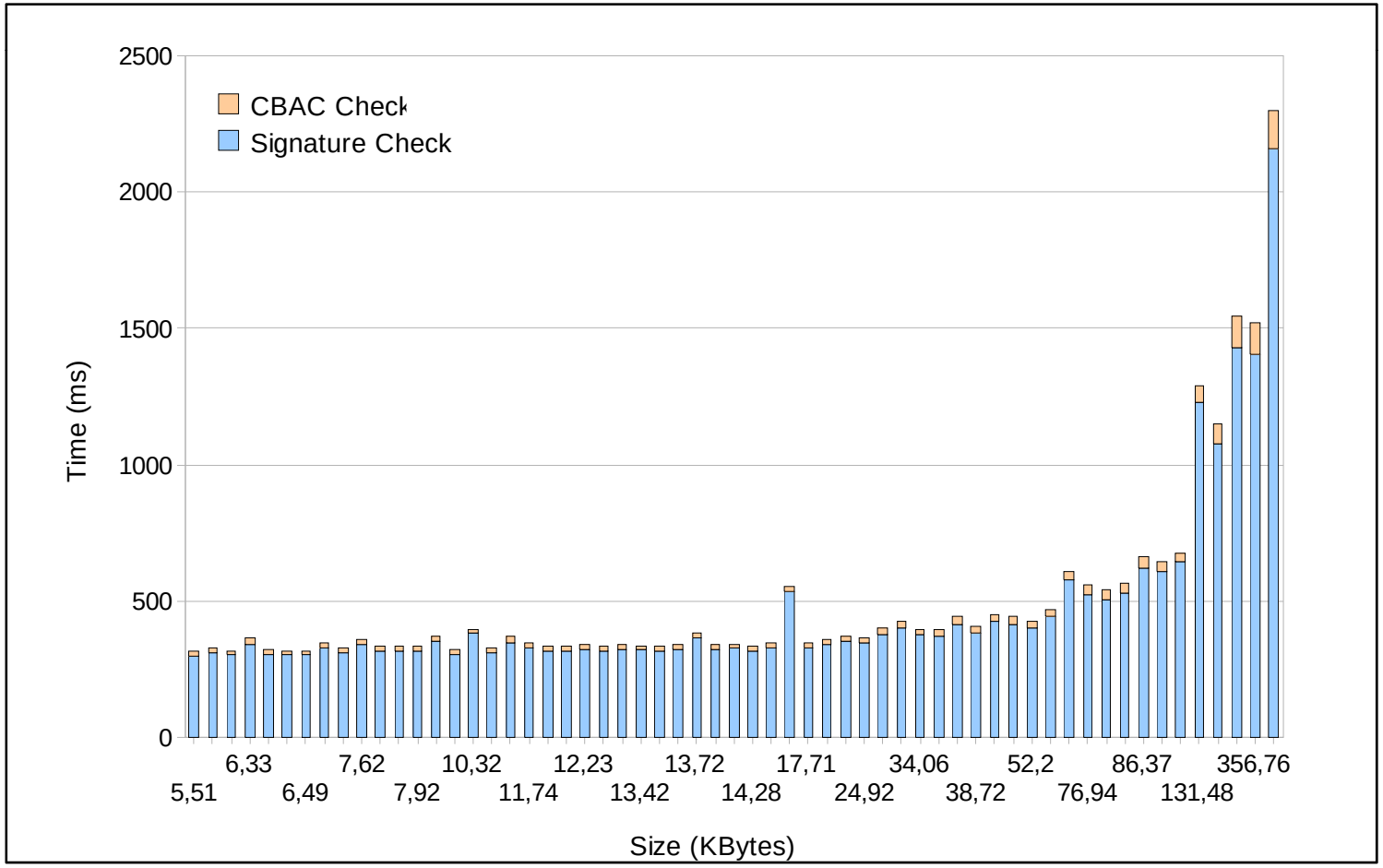
Enforcing Security for Components: CBAC, WCA

- The CBAC model: Definition



Enforcing Security for Components: CBAC, WCA

- The CBAC model: Performances



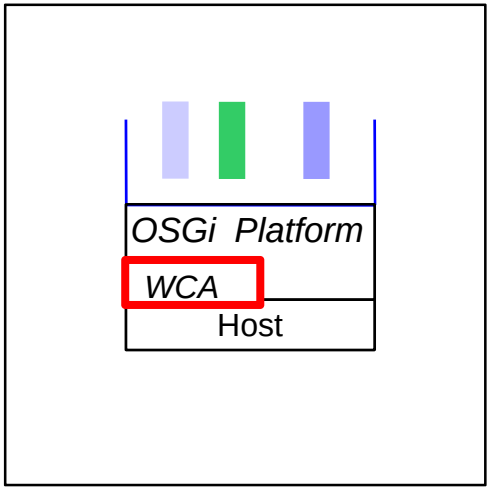
Enforcing Security for Components: CBAC, WCA

- The CBAC model: Benefits and limitations
  - Benefits
    - No runtime overhead, reduced install time overhead
    - No application interruption, at the cost of false positive
    - No misleading pop-up windows
    - Arbitrary methods and meta-data can be set as sensitive
      - Enables to protect against vulnerabilities that are discovered after design
    - Protection Rate: 50 % for the ‘Malicious Bundles’ catalog entries
  - Limitations
    - Policies must be defined in advanced

Enforcing Security for Components: CBAC, WCA

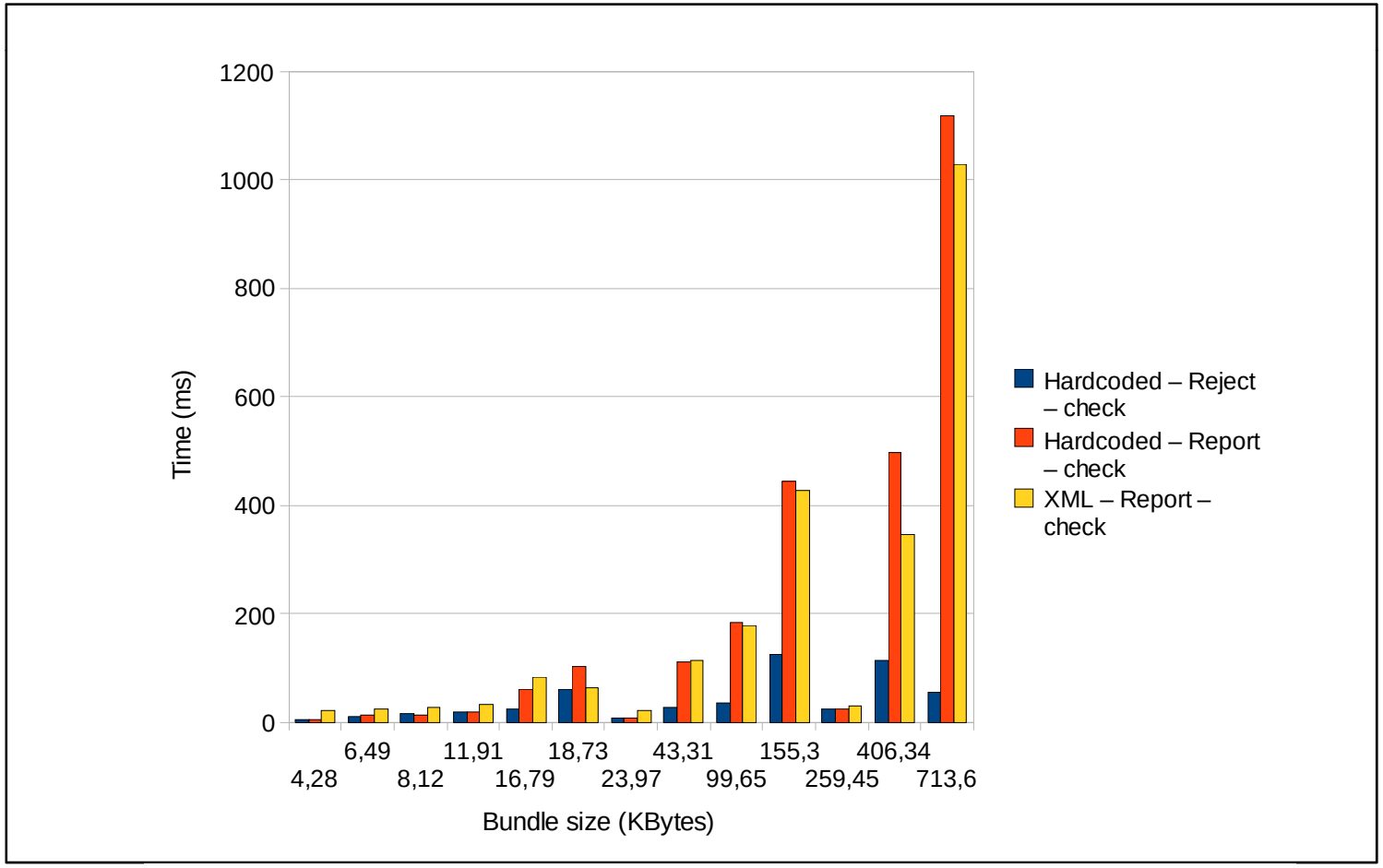
- The WCA approach: Principles
  - **W**eak **C**omponent **A**nalysis
  - Goal
    - Prevent issues from the ‘Vulnerable Bundles’ catalog
  - Principles
    - Vulnerability identification through static analysis
      - In exposed code only
      - Through the code meta-model
      - Matching with ‘vulnerability patterns’
    - Development and install time use
      - XML version for flexibility
      - Hardcoded version for performance

Enhancing Automated Detection of Vulnerabilities in Java Components, International Conference on Availability, Reliability and Security (AReS 2009).



Enforcing Security for Components: CBAC, WCA

- The WCA approach: Performances



Enforcing Security for Components: CBAC, WCA

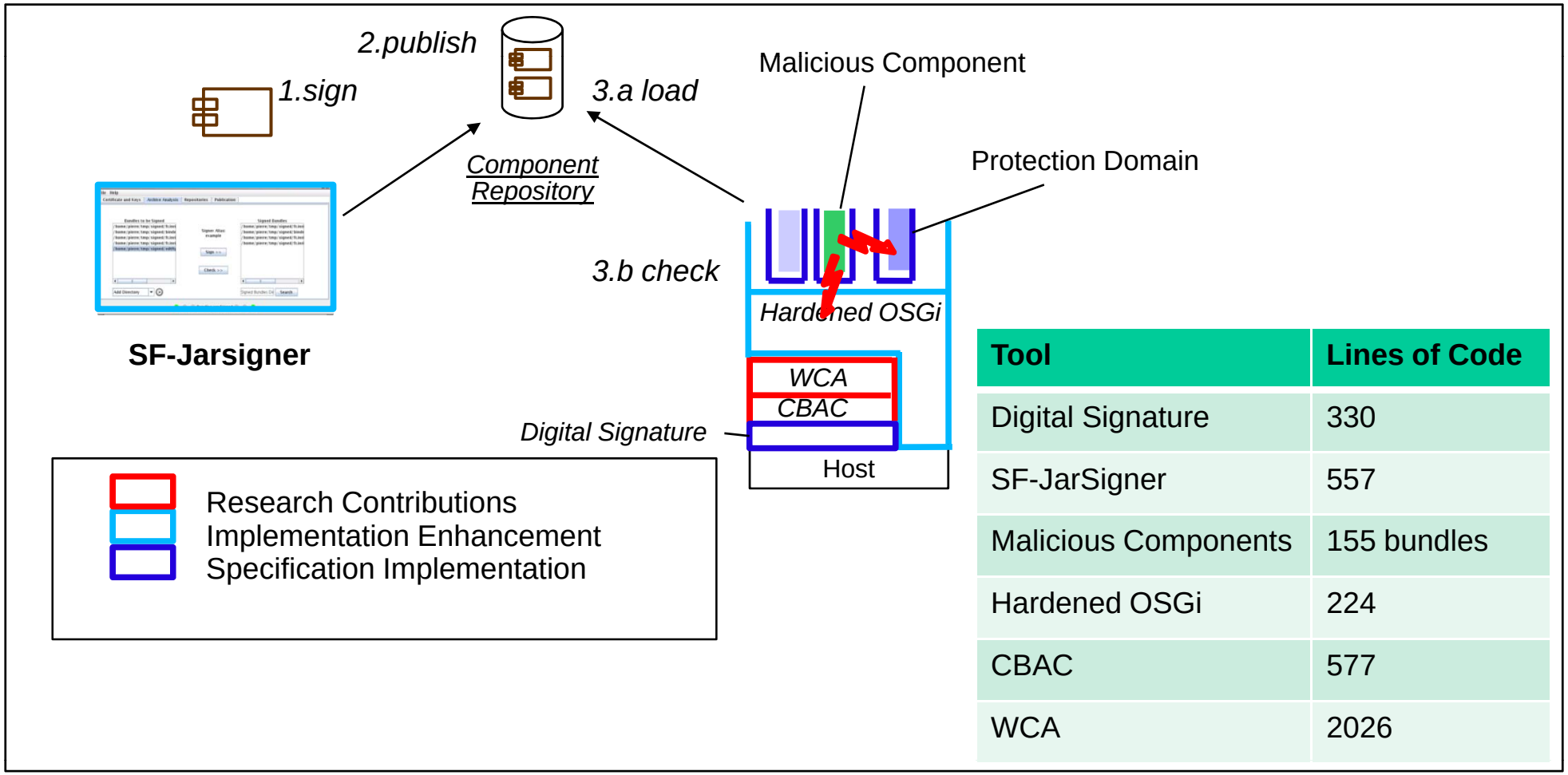
- The WCA approach: Benefits and limitations
  - Benefits
    - Identification of exploitable vulnerabilities in Java components
    - According to the exposition of the code
    - Principally easy to extend
    - Development and runtime use
    - Protection Rate: 36 % for the ‘Vulnerable Bundles’ catalog entries
  - Limitations of the implementation
    - Hardcoded version is slower
    - Only structural patterns are supported so far
    - Limited flexibility of the definition of patterns

Outline

- Security for Java-based Software Systems
- Contributions
- **Conclusions**

Conclusions

- Development overview



Tool	Lines of Code
Digital Signature	330
SF-JarSigner	557
Malicious Components	155 bundles
Hardened OSGi	224
CBAC	577
WCA	2026

Conclusions

- Evaluation of the proposed solutions
  - *SPIP*
    - Promising methodology for security analysis
    - Requires
      - Validation on further systems
      - Support for cost estimation
  - Tools for secure component-based applications
    - CBAC
      - Refined static analysis approach
    - WCA
      - Only a subset of best practices are enforced so far
      - Need of actual isolation between the bundles
  - Consider further attack vectors

Conclusions

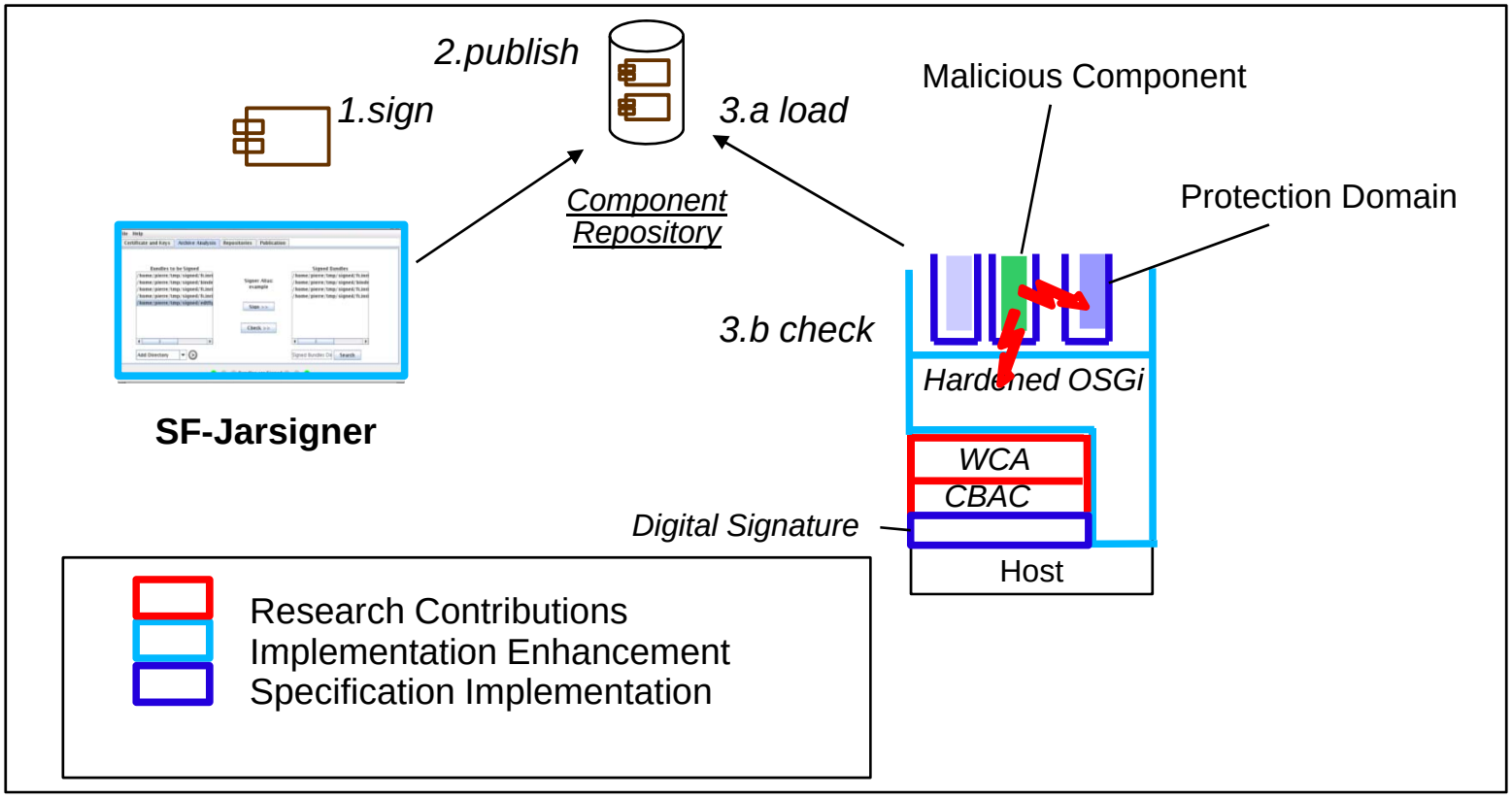
- Who can benefit from this work ?

Role	Platform developer	Application architect	Application developer
Focus on	Execution environment	Architecture	Components
System entity	Platform	Components	Components
Life-Cycle Activity	Platform design and coding	Application design	Application Coding
	Security analysis for the Platform	*Security analysis for all*	*Security analysis for the Components*
Our propositions	*Hardened OSGi*	*CBAC*	*WCA*
		Integration	

Conclusions

- Open challenges
  - Resource isolation
    - First solution: Integration with the JnJVM
  - Development for industrial use of the OSGi platform
    - Specifications
    - Life-cycle support for bundles
    - Management
    - Critical applications: strong isolation between applications
    - Multi-user applications: strong access control mechanism

Questions ?


2.publish
1.sign
3.a load
Malicious Component
Component Repository
Protection Domain
3.b check
SF-Jarsigner
Hardened OSGi
WCA
CBAC
Digital Signature
Host
Research Contributions
Implementation Enhancement
Specification Implementation

References

- Karen M. Goertzel, Thoedore Winograd, Holly L. McKinley, Lyndon Oh, Michael Colon, Thomas Mcibbon, Elaine Fedchak, and Robert Vienneau. *Software Security Assurance: a State-of-The-Art Report (SOAR). Information A*ssurance Technology Analysis Center (IATAC) and Data and Analysis Center for Software (DACS), July 2007.
- Michael Howard, Jon Pincus, and Jeanette M. Wing. *Computer Security in the 21st Century, chapter Measuring Relative Attack Surfaces, pages 109–137.* Springer, March 2005.
- Capers Jones. *Applied software measurement: assuring productivity and quality.* McGraw-Hill, Inc., New York, NY, USA, 1999.
- Yvan Royon, *Environnements d'exécution pour passerelles domestiques*, PhD Thesis, INSA-Lyon, December 2007.
- Noha Ibrahim, *Spontaneous Integration of Services in Pervasive Environments*, PhD Thesis, INSA-Lyon, September 2008.

Selected Publications

- ***Journal Article***
- Security Benchmarks of OSGi Platforms: Toward Hardened OSGi, Pierre Parrend, Stephane Frénot, Software, Practice and Experience. Accepted for publication (September 2008).
- ***International Conferences, Industrial Conferences***
- Enhancing Automated Detection of Vulnerabilities in Java Components, Pierre Parrend, Forth International Conference on Availability, Reliability and Security (AReS 2009), Fukuoka, Japan, 16th – 19th March 2009.
- Classification of Component Vulnerabilities in Java Service Oriented Programming (SOP) Platforms, Parrend, Stéphane Frenot, Conference on Component-based Software Engineering, Karlsruhe, Germany, 14-17 October 2008.
- Component-based Access Control: Secure Software Composition through Static Analysis, Pierre Parrend, Stéphane Frenot, Software Composition, Budapest, Hungary, 29-30 March 2008.
- Multi-service, Multi-protocol Management for Residential Gateways Home Network Management, Y. Royon, P. Parrend, S. Frénot, S. Papastefanos, H. Abdelnur, D. Van de Poel, S. Frenot, BB Europe, Antwerp, December 3-6, 2007.
- ***Research Reports***
- More Vulnerabilities in the Java/OSGi Platform: a Focus on Bundle Interactions, Pierre Parrend, Stephane Frenot, INRIA Research Report n°6649, September 2008.
- Java Components Vulnerabilities - An Experimental Classification Targeted at the OSGi Platform, Pierre Parrend, Stéphane Frenot, INRIA Research Report n° 6231, June 2007´.